MelbourneDAC 2003

Undiscovered Worlds – Towards a Framework for Real-Time Procedural World Generation

Stefan Greuter, Jeremy Parker

RMIT Centre for Animation and Interactive Media

stefan.greuter@gmx.de, jeremy.parker@rmit.edu.au

Nigel Stewart, Geoff Leach

RMIT School of Computer Science and IT

nigels@nigels.com, gl@cs.rmit.edu.au

ABSTRACT: With advances in computer hardware, 3D game worlds are becoming larger and more complex. Consequently the development of game worlds becomes increasingly time and resource intensive. This paper presents a framework for generation of entire virtual worlds using procedural generation. The approach is demonstrated with the example of a virtual city.

KEYWORDS: Games Culture, Games System Design, Games Theory, Interactive Architecture, MUDs, Virtual Reality, Virtual Worlds

INTRODUCTION

The visual quality of three-dimensional, polygon based computer games improves with each new generation of graphics hardware. In fact the processing speed of graphic hardware doubles every year. Each new generation of computer hardware facilitates computer games that feature game worlds in higher detail and larger extent. All object geometries and textures in a game world need to be created in a process that can be very time consuming. As game worlds grow, more time and resources are required for their creation. Where do we go from here?

Approaches to minimize the costs for the creation of game worlds include reusing textures and geometries for objects that occur frequently throughout the game, such as trees and crates. However, the repetitive use of artwork is often obvious to the user, and counteracts the verisimilitude of a believably realistic virtual world. Another approach is to generate recurring objects or even entire game worlds in a so-called procedural approach, which not only adds a controllable randomness to the virtual world but also yields outcomes that are unexpected in form and appearance.

This paper presents a framework for the real-time procedural generation of virtual worlds, and is aimed at a non-technical audience. The virtual worlds are self contained and neither import nor export data. The worlds can expand to an extent that would require many lifetimes to explore, which we term 'pseudo infinite' [1]. The framework is suitable for real-time virtual worlds running on commodity hardware and has been prototyped with the example of a virtual city. A more technical description of our work [1] appeared recently at Graphite 2003.

Related Work

Previous research into the generation of plants [2], trees [3] and cities [4] has demonstrated that a variety of objects of the same type but with a different visual appearance, can be generated using a procedural approach. Such objects show no apparent repetition and create a much more realistic picture of the randomness and variety that exists in our real world.

Procedural modelling of natural phenomena such as plants and terrains has been of great interest to researchers. However, other areas such as procedural generation of manmade structures such as walls and buildings, is less common. Few applications use generated procedural geometry and textures to create entire virtual worlds. Two example applications are Intel's procedural world demo [5] and Virtual Gardening [6].

Intel's procedural world demo [5] shows procedurally generated terrain with trees and two dimensional cloud layer based on Perlin noise [7]. The terrain can be freely explored on the horizontal plane in real-time and expands around the user's point of view.

Virtual Gardening [6] is an application which features the interactive design of a virtual garden environment. The program simulates the growth of plants and trees over time, influenced by changing seasons.

FRAMEWORK COMPONENTS

Our framework for real-time procedural world generation is illustrated in Figure 1. It consists of three major components: view frustum filling, geometry caching and geometry generation. View frustum filling constrains procedural generation of geometry to regions visible from the viewpoint. Geometry caching stores a temporary database of previously generated geometry. The building geometry generator is a collection of procedures for constructing particular types of objects.

View Frustum Filling

A virtual world can be seen as a collection of discreet three-dimensional objects. During navigation however, only a fraction of the entire virtual world is visible to the user. *View frustum filling* determines the visibility of object-spaces within the cameras viewing volume. Each visible object-space is evaluated for its object-type; then the object can be generated.

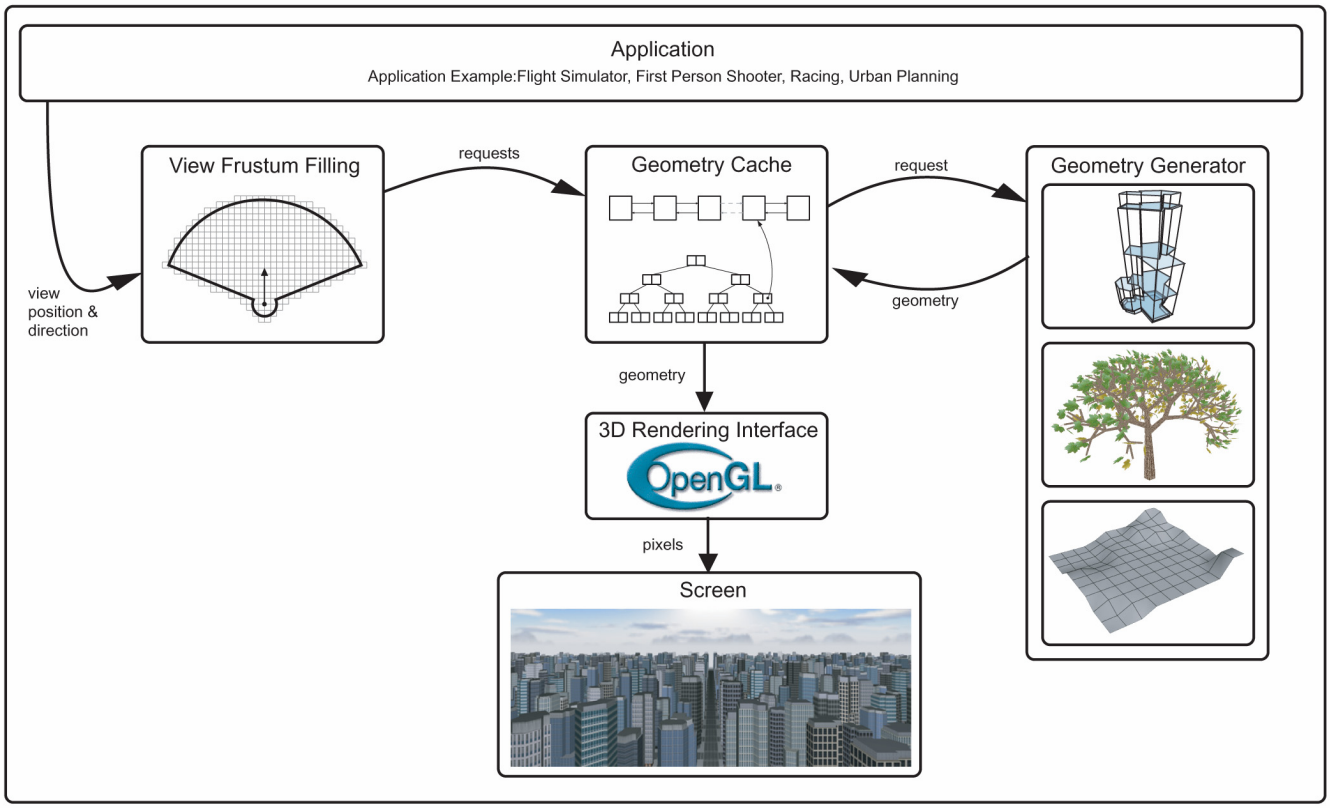

Figure 1: Framework Architecture

View frustum culling is a similar technique, commonly used in computer games to restrict rendering to potentially visible geometry. The alternative proposed in this framework is to constrain procedural generation to potentially visible regions, rather than culling pre-existing geometry.

Caching

While the user is navigating through the world only a few new visible objects appear on the horizon. The geometry of most other objects remains the same. A caching data structure facilitates the use of already generated objects in memory. In subsequent frames cached objects can be drawn without regenerating them. We reported previously that caching is faster, by a factor of approximately 8 [1], than regenerating all objects on the screen for every frame. The object remains in memory until it is no longer needed. The cache keeps track of all objects and removes the least recently used objects from memory. Visible objects which are not available in the cache need to be generated (or re-generated) by the geometry generator.

Geometry Generator

The procedural geometry generator is a collection of procedural object-types that produces outcomes that are repeatable but not identical. The generation of each object-type is based on one seed value and optional parameters. By changing the seed value and parameters a variety of building geometries of different height, width and depth can be generated. However, the objects that the algorithm generates will always be a variation of a certain object-type, which is subject to its own set of generation rules. Hence an office building generator can only generate a variety of office buildings in within the boundaries set by the specifications. The final outcome of the generation, however, can not be predicted and is sometimes even unexpected.

PROCEDURAL CITIES

Our framework has been implemented using the example of a simple virtual city consisting of a regular street grid and skyscraper type buildings. The visual appearances of the generated buildings are inspired by the shape and diversity of buildings in Melbourne. The generated city is extremely large, consisting of $4x10^{18}$ geometrically different buildings; about 600,000,000 times the world's current human population. The entire city can be freely explored from a first person perspective. One of our virtual cities is illustrated in Figure 2.

City Consistency

The generation of objects and their properties is based on a global *citySeed* and the objects position in within the city. Hence it is possible to re-generate the identical building on a previously position visited A similar system has been outlined by Lecky-Thompson [8].

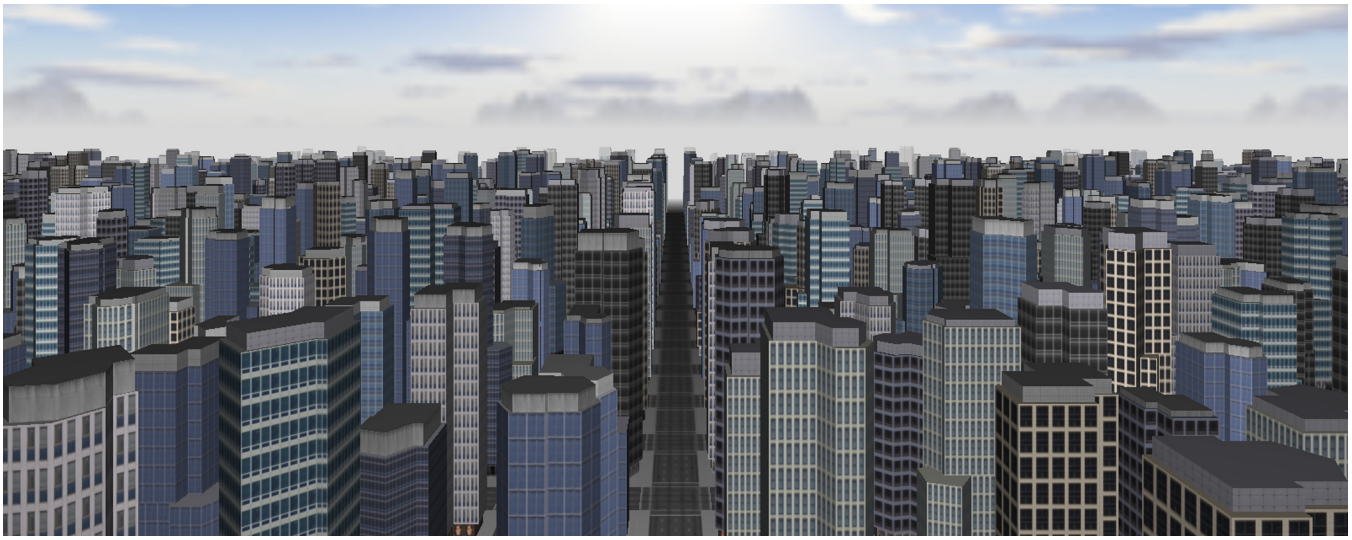

Figure 2: Real Time Procedural City

Building Floor Plan Generation

In the virtual city we used an iterative function system to generate floor plans of buildings, which are extruded to building facades. Each building consists of several floor plans. Floor plans are two-dimensional polygons that are composed of regular polygons and rectangles in an iterative process [1]. The floor plan generation process is illustrated in Figure 3.

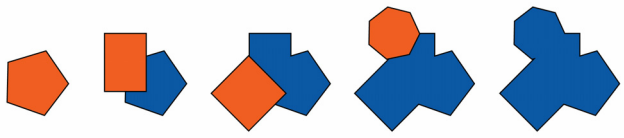

Figure 3: Floor Plan Generation

Building Facade Generation

The facade of each building is composed of several steps. Each step consists of a floor plan that is extruded by an arbitrary height. The extrusion process starts with the top floor. Every consecutive step consists of a floor plan that adds at least one more polygon. The final extruded step consists of the most complex floor plan. The building extrusion is illustrated in Figure 4. The virtual city uses a limited set of textures, which in principle could also be procedurally generated.

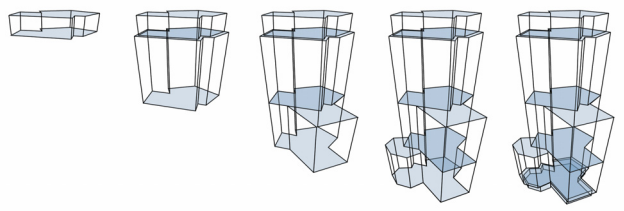

Figure 4: Building Facade Generation

DISCUSSION

The proposed framework is not limited to generation of only virtual cities. By changing parameters such as scale, range of variety, generation algorithms and textures, places can be generated that are have nothing in common with a virtual city. The framework merely provides the foundation on which three dimensional virtual worlds can be built without the need to model objects and variations of objects manually.

Many computer game levels can not be completed in the first attempt; they need to be replayed several times. Repeated playing of the same static level, however, exposes the player to an already explored and therefore familiar environment, which typically offers few if any, new or surprising elements. Game levels that are generated in a procedural approach offer more variety and extent to stimulate players with new and exciting content to explore. A new game world could be generated by the proposed framework every time the player starts a new game. As the level content is generated as needed, even areas that are 'off the beaten track' can be explored. Potentially, the game engine can monitor a player's performance and adjust the level of difficulty to maintain the challenge of the game and increase its longevity.

Research on procedural generation of varying three-dimensional content could also benefit other disciplines. The findings of this research could provide an inspirational tool for designers and architects wishing to explore and combine various forms of generated structures. Figure 5 illustrates an example of the variety that can be achieved using procedural building generation in varying levels of complexity. Flight-simulators, for example, could confront pilots with procedurally generated training scenarios that challenge the pilot's abilities and rapid decision-making skills. The framework could be extended to mix real data with generated data. An urban planning simulation could provide a view of a

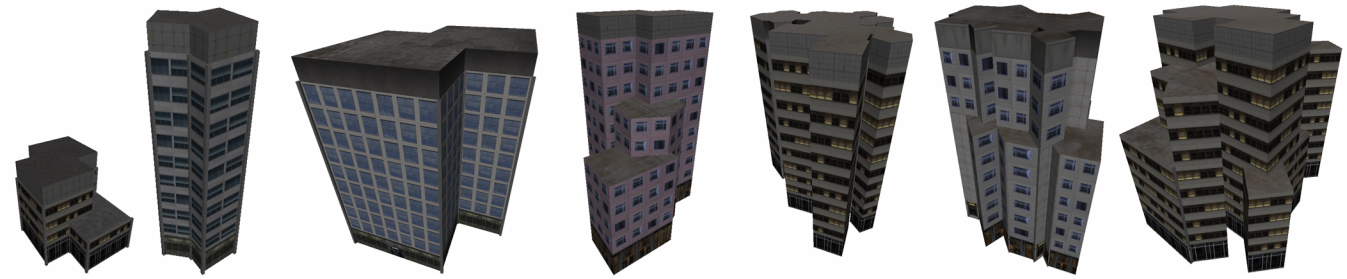

Figure 5: Procedural Variety

particular location in a city using existing geometric data while backdrop objects are generated. The framework could also be used by the lay person to explore their own creations of virtual worlds without the need to design each individual building.

CONCLUSION AND FURTHER WORK

We have proposed a framework for the procedural generation of entire virtual worlds in real-time. In contrast to commercial game engines, our approach does not rely on virtual worlds which are stored on disk or transmitted over the Internet. The generated worlds can be extremely large in extent without repetitive use of identical objects. As in the real world, objects can appear to be similar but they are not identical.

The framework has been demonstrated with the example of a virtual city. The city is a self-contained space and is populated with a simple street grid and generated building structures with various geometries. The city can be explored in real-time at interactive frame rates on consumer PCs with graphics hardware.

The proposed framework poses many problems and opportunities, both technically and creatively.

From a technical point of view, our implementation does not include collision detection. There are no laws of physics preventing the user from flying through a building. Secondly, the framework does not track building changes such as damage or aging.

On the creative side the framework requires more procedural object-types to generate virtual worlds other than cities. Such object types could be plants, trees and terrain. Especially the procedural generation of plants, trees and terrain are well researched areas which would complement the framework and make the generation of virtual worlds of a more natural character, like forests, mountain ranges or tropical islands possible. Although we focussed in our demonstration on a more ‘realistic’ representation of a virtual world, the framework is not limited to only generate ‘realistic’ looking 3D spaces. By adding custom made static or procedural objects the framework can also generate a variety of abstract forms which can be explored in real time.

In essence a user could create themselves may kinds of virtual worlds with this framework. By selecting the generation algorithms and by manipulating the parameters that drive the generation process the world can be populated with objects that are generated, rather than creating all objects manually. By changing the parameters, the user can enter in a world that looks different every time and that goes on virtually forever.

ACKNOWLEDGMENTS

This research is partially funded by a scholarship from the German Academic Exchange Service (DAAD) and an International Postgraduate Research Scholarship (IPRS) from the Australian government and RMIT University.

REFERENCES

1. Greuter, S., Parker, J., Stewart, N., et al. Real-time Procedural Generation of 'Pseudo Infinite' Cities, in Proc. GRAPHITE 2003, (Melbourne, 2003) ACM SIGGRAPH pp. 87-94.

2. Prusinkiewicz, P., Lindenmayer, A., Hanan, J. S., et al. The Algorithmic Beauty of Plants. Springer, New York, 1990

3. Oppenheimer, P. E. Real Time Design and Animation of Fractal Plants and Trees, in Proc. ACM SIGGRAPH, (Dallas, 1986) ACM pp. 55 - 64.

4. Parish, Y. I. H. and Müller, P. Procedural Modelling of Cities, in Proc. ACM SIGGRAPH, (Los Angeles, 2001) ACM Press pp. 301 - 308.

5. Macri, D. and Pallister, K. "Procedural 3D Content Generation." (2001) http://cedar.intel.com/ Access: 03.02.2001

6. JFP Inc. "Virtual Gardening." (1999) http://www.jfp.co.jp/garden_e/ Access: 12.11.2002

7. Perlin, K. An Image Synthesizer, in Proc. ACM SIGGRAPH, (San Francisco, 1985) pp. 287-296

8. Lecky-Thompson, G. W. Infinite Game Universe: Mathematical Techniques. Charles River Media, Hingham, Massachusetts, 2001